

**..** OBJECTIF DOCUMENT

Le document d'évaluation de la conformité (Compliance Assessment) a pour objectif de vérifier que l'architecture logicielle et les composants de notre projet Your Car Your Way respectent les normes et les meilleures pratiques en matière de développement logiciel, de sécurité, de gestion des données et d'infrastructure. Cette évaluation est essentielle pour garantir la fiabilité, la sécurité et la performance de notre solution. Ce document reprend les éléments du cadre architectural présenté dans le cahier des charges fonctionnel (Architecture Definition Document) et applique une liste de contrôles spécifiques pour chaque composant identifié.

**..** Description générale de l’architecture

L'architecture de Your Car Your Way se compose de plusieurs entités principales telles que les clients, les employés, les véhicules, les agences, et les locations. Chacune de ces entités est représentée par une table dans notre base de données, et les relations entre ces entités sont soigneusement définies pour respecter les principes de normalisation, notamment la Troisième Forme Normale (3NF). L'architecture est conçue pour être modulaire et extensible, avec une séparation claire des responsabilités entre les différentes couches de l'application.

**..** Liste de contrôle de l’architecture terminée

# Composants logiciels

Les composants logiciels propres au projet ont été développés en respectant les normes de codage et les bonnes pratiques de développement.

1. **Frontend (FE) :**

* **Technologie :** Angular
* **Description :** L'interface utilisateur est développée sous la forme d'une Single Page Application (SPA) utilisant Angular. Cette technologie permet de créer des composants réutilisables, une gestion efficace de l'état de l'application, et une navigation fluide sans rechargement complet de la page.
* **Bonne Pratique :** Utilisation de TypeScript pour des types stricts, tests unitaires avec Jasmine/Karma, et respect des standards d'accessibilité.

1. **Backend (BE) HTTP :**

* **Technologie :** Java avec Spring Boot
* **Description :** Le serveur backend pour les requêtes HTTP est développé en Java en utilisant le framework Spring Boot, qui permet une gestion efficace des dépendances, une configuration simplifiée, et une robustesse accrue pour les services métiers.
* **Bonne Pratique :** Respect des principes SOLID, utilisation des annotations Spring pour la gestion des transactions et la sécurité, tests unitaires avec JUnit et Mockito, et intégration continue avec Jenkins.

1. **Backend (BE) WebSockets :**

* **Technologie :** Node.js avec Socket.IO
* **Description :** Le backend pour les requêtes WebSockets est développé en Node.js en utilisant Socket.IO, qui facilite la communication en temps réel entre le client et le serveur. Cette technologie permet d'établir des connexions bidirectionnelles efficaces, tout en gérant la scalabilité et la performance des échanges de données.
* **Bonne Pratique :** Respect des meilleures pratiques de développement JavaScript, utilisation de l'authentification JWT pour sécuriser les connexions, gestion des erreurs et des événements, tests unitaires avec Jest, et mise en place de pipelines CI/CD pour une intégration et un déploiement continus.

1. **Base de Données (BDD) :**

* **Technologie :** PostgreSQL
* **Description :** La base de données relationnelle PostgreSQL est utilisée pour le stockage des données. Les entités sont normalisées selon la Troisième Forme Normale (3NF) pour éviter les redondances et garantir l'intégrité des données.
* **Bonne Pratique :** Utilisation des index pour optimiser les requêtes, sauvegardes régulières, et chiffrage des données sensibles.

1. **Persistance de l'Authentification :**

* **Technologie :** JSON Web Tokens (JWT)
* **Description :** La persistance de l'authentification est assurée par l'utilisation de JSON Web Tokens (JWT). Les tokens JWT sont générés lors de l'authentification réussie et sont utilisés pour authentifier les requêtes des utilisateurs par la suite.
* **Bonne Pratique :** Utilisation de JWT signés et chiffrés, stockage sécurisé des tokens côté client (par exemple, dans des cookies HTTPOnly), et mise en place de mécanismes de rafraîchissement des tokens pour prolonger les sessions de manière sécurisée.

1. **Balancing des Requêtes :**

* **Technologie :** Elastic Load Balancer (ELB) d'AWS
* **Description :** L'équilibrage de charge des requêtes est réalisé à l'aide d'Elastic Load Balancer (ELB) d'AWS. Cette configuration permet de répartir les requêtes entrantes sur plusieurs instances du backend pour améliorer la disponibilité et la performance de l'application.
* **Bonne Pratique :** Configuration d'ELB pour la gestion des sessions, surveillance des performances des serveurs backend, et mise en place de mécanismes de failover.

# Services ou composants tiers logiciels

Cette section documente les services tiers utilisés dans le projet, y compris les communications WebSocket (WS) pour le chat, la visioconférence via WebRTC, ainsi que les échanges CRUD entre le front-end (FE) et le back-end (BE) via HTTPS, et les interactions entre le back-end (BE) et la base de données (DB) utilisant Spring Data JPA.

Les services ou composants tiers logiciels utilisés dans le cadre du projet Your Car Your Way ont été soigneusement sélectionnés pour répondre aux besoins spécifiques de l'application notemment avec l'architecture tout en garantissant la sécurité, la performance et la compatibilité avec notre architecture globale.

1. **Chat via WebSocket**

* **Description :** WebSocket est utilisé pour la communication en temps réel entre les utilisateurs de l'application, facilitant le chat instantané. Contrairement aux protocoles HTTP traditionnels, WebSocket permet une communication bidirectionnelle et persistante entre le client (front-end) et le serveur (back-end), ce qui est essentiel pour des fonctionnalités telles que les notifications en temps réel et les salles de discussion.
* **Bonne Pratique :** Implémentation de mécanismes de sécurité tels que l'authentification des utilisateurs, utilisation des WebSocket Secure et la gestion des autorisations pour prévenir les attaques par injection ou l'accès non autorisé aux canaux de communication.

1. **Visioconférence via WebRTC**

* **Description :** WebRTC est utilisé pour la mise en œuvre de la visioconférence, permettant une communication audiovisuelle en temps réel directement dans le navigateur web sans nécessiter de plugins tiers. Il facilite la transmission de flux multimédias (audio, vidéo, données) entre les participants, garantissant une expérience utilisateur fluide et efficace pour les réunions virtuelles.
* **Bonne Pratique :** Mise en place de mesures de sécurité telles que le chiffrement des médias et la gestion des identités des utilisateurs pour assurer la confidentialité des données échangées lors des visioconférences.

1. **Échanges CRUD entre le front-end (FE) et le back-end (BE) via HTTPS**

* **Description :** Les échanges CRUD (Create, Read, Update, Delete) entre le front-end et le back-end sont sécurisés via HTTPS. Ce protocole de communication sécurisé utilise le chiffrement TLS (Transport Layer Security) pour assurer la confidentialité et l'intégrité des données transitant entre le client (FE) et le serveur (BE). Il protège également contre les attaques de type man-in-the-middle et garantit l'authenticité des serveurs.
* **Bonne Pratique :** Configuration correcte des certificats SSL/TLS, gestion sécurisée des sessions utilisateur, et validation rigoureuse des entrées utilisateur pour prévenir les failles de sécurité telles que les injections SQL ou les attaques XSS (Cross-Site Scripting).

1. **Échanges entre le back-end (BE) et la base de données (DB) en HTTP avec Spring Data JPA**

* **Technologie :** Spring Data JPA (Java Persistence API)
* **Description :** Spring Data JPA est utilisé pour la persistance des données dans la base de données relationnelle. Il simplifie le développement d'applications en utilisant des annotations pour définir des entités et des requêtes, facilitant ainsi l'accès et la manipulation des données à travers une couche d'abstraction.
* **Bonne Pratique :** Utilisation de transactions pour assurer la cohérence des opérations de base de données, gestion efficace du pool de connexions pour optimiser les performances, et implémentation de pratiques de sécurité telles que l'échappement des paramètres pour prévenir les attaques par injection SQL.

1. **Échanges en temps réel entre le front-end (FE) et le back-end (BE) en WebSockets**

* **Technologie :** Socket.io
* **Description :** Les échanges en temps réel entre le front-end et le back-end sont réalisés via WebSockets, en utilisant Socket.IO pour établir des connexions bidirectionnelles. Cela permet d’envoyer et de recevoir des données instantanément, idéal pour les fonctionnalités nécessitant une mise à jour en temps réel, comme les notifications, les messages texte avec ou sans fichier, ou encore à mettre à jour l'UI de l'application sans avoir besoin de rafraîchir la page.
* **Bonne Pratique :** Utilisation de Socket.IO pour gérer les connexions, mise en place d'une authentification sécurisée avec JWT passée après le handshake, validation des entrées pour éviter les vulnérabilités, et gestion des événements pour une communication fluide.

1. **Échanges entre le back-end (BE) et la base de données (DB) en HTTP avec Spring Data JPA**

* **Technologie :** Sequelize (ORM)
* **Description :** Sequelize est utilisé comme ORM (Object Relational Mapper) pour interagir avec la base de données PostgreSQL. Il facilite la gestion des modèles de données, des migrations et des requêtes complexes, tout en offrant une interface intuitive pour la manipulation des données.
* **Bonne Pratique :** Mise en œuvre de migrations pour gérer les schémas de la base de données, utilisation de transactions pour assurer la cohérence des opérations de données, et validation des entrées pour prévenir les injections SQL. De plus, il est essentiel de gérer le pool de connexions pour optimiser les performances des requêtes.

# Gestion des données

Sur la gestion des données, nous mettrons l'accent sur PostgreSQL 16, les normes 3NF et les échanges encryptés via HTTPS :

1. **Système de Gestion de Base de Données (SGBD) :** PostgreSQL 16

* **Description :** PostgreSQL 16 est utilisé comme SGBD principal pour le stockage et la gestion des données de l'application. Il offre une architecture robuste de base de données relationnelle, prise en charge des transactions ACID (Atomicité, Cohérence, Isolation, Durabilité), ainsi que des fonctionnalités avancées telles que la réplication, le partitionnement et le support JSONB.
* **Bonne Pratique :** Configuration appropriée des paramètres de sécurité et de performance de PostgreSQL, y compris l'authentification forte, le chiffrement des données sensibles au repos, et la sauvegarde régulière des données pour assurer la résilience et la disponibilité.

1. **Normes de Normalisation 3NF (Troisième Forme Normale) :**

* **Description :** Les données sont structurées selon tous les 3 principes de la troisième forme normale (3NF), qui vise à réduire la redondance des données et à améliorer l'intégrité des données en évitant les dépendances transitives. Les trois normes de normalisation sont :

1. **1NF (Première Forme Normale) :** Chaque colonne d'une table doit contenir des valeurs de type primitif (pas d'objets, de tableaux ou de chaînes contenant plusieurs valeurs). Cela garantit que chaque cellule contient une seule valeur.
2. **2NF (Deuxième Forme Normale) :** Les tables de jointure n-à-n, qui utilisent des clés primaires composées, doivent contenir des colonnes qui expriment des propriétés de la relation entre les tables. Par exemple, si nous souhaitions joindre la table Clients et la table Agencies, nous créerions une table de jointure appelée ClientAgencies. Cette table contiendrait les colonnes suivantes :
   * **ClientID :** clé étrangère référant à la table Clients.
   * **AgencyID :** clé étrangère référant à la table Agencies.
   * **DateInscription :** une colonne supplémentaire qui indique la date à laquelle le client s'est inscrit auprès de l'agence.
   * Ainsi, chaque enregistrement dans la table ClientAgencies représenterait une relation unique entre un client et une agence, tout en conservant les informations spécifiques à cette relation.
3. **3NF (Troisième Forme Normale) :** Les colonnes non-clés d'une table doivent être indépendantes les unes des autres. Elles doivent dépendre uniquement de la clé primaire et non d'autres colonnes non-clés. Par exemple, si une table contient des colonnes pour le prénom et le nom, le prénom ne doit pas dépendre d'une autre colonne comme le nom.

* Bonne Pratique : Validation continue du schéma de base de données pour maintenir la conformité aux normes 3NF, en veillant à ce que les nouvelles entités et relations soient correctement normalisées lors de l'évolution de l'application.

1. **Échanges encryptés via HTTPS :**

* **Description :** Tous les échanges de données sensibles entre le front-end (FE) et le back-end (BE), ainsi qu'entre le BE et la base de données (DB), sont sécurisés via HTTPS. Ce protocole garantit le chiffrement des données en transit, assurant ainsi la confidentialité et l'intégrité des informations échangées entre les composants de l'architecture.
* **Bonne Pratique :** Utilisation de certificats SSL/TLS valides, configuration correcte des paramètres de sécurité du serveur web pour activer HTTPS, et audit régulier des protocoles de sécurité pour détecter et corriger les éventuelles vulnérabilités.

1. **Échanges encryptés via WSS :**

* **Description :** Tous les échanges de données sensibles entre le front-end (FE) et le back-end (BE) via WebSockets sont sécurisés en utilisant le protocole WSS (WebSocket Secure). Ce protocole garantit le chiffrement des données en transit, assurant ainsi la confidentialité et l'intégrité des informations échangées. L'authentification des utilisateurs peut être réalisée lors de l'établissement de la connexion, souvent via des tokens JWT.
* **Bonne Pratique :** Utilisation de WSS pour toutes les connexions WebSocket, mise en place d'une authentification sécurisée (ex. : JWT) durant le handshake, validation rigoureuse des données entrantes pour prévenir les attaques telles que les injections ou les attaques de type Cross-Site WebSocket Hijacking, et gestion des événements de connexion et de déconnexion pour maintenir une communication stable.

# Infrastructure

Pour la section sur l'infrastructure, nous mettrons en avant AWS, ELB, ainsi que la dockerisation et le déploiement des composants front-end (FE) et back-end (BE) dans des instances EC2 avec une instance RDS pour PostgreSQL:

1. **Environnement de Déploiement :** AWS (Amazon Web Services)

* **Description :** L'application est déployée dans un environnement cloud AWS, utilisant des services cloud pour le calcul, le stockage, et la gestion de l'infrastructure.
* **Bonne Pratique :** Utilisation des bonnes pratiques de sécurité AWS, comme la configuration des groupes de sécurité, le chiffrement des données en transit et au repos, ainsi que la gestion des accès IAM pour limiter les permissions selon le principe du moindre privilège.

1. **Elastic Load Balancer (ELB) :**

* **Description :** ELB est utilisé pour l'équilibrage de charge des requêtes entrantes vers les instances EC2 du front-end (FE) et du back-end (BE). Cela permet de distribuer la charge et d'améliorer la disponibilité et la résilience de l'application.
* **Bonne Pratique :** Configuration d'ELB pour supporter HTTPS, gestion des sessions, et intégration avec AWS Auto Scaling pour ajuster automatiquement la capacité en fonction du trafic.

1. **FE et BE Dockerisés dans des instances EC2 :**

* **Description :** Les composants front-end (FE) et back-end (BE) sont conteneurisés avec Docker et déployés dans des instances Amazon EC2. Cette approche offre une gestion flexible et efficace des applications, avec la possibilité de scaller horizontalement selon les besoins.
* **Bonne Pratique :** Utilisation de Docker pour l'isolation des environnements d'exécution, gestion des versions de conteneurs, et automatisation du déploiement à l'aide de services comme Amazon ECS (Elastic Container Service) ou Amazon EKS (Elastic Kubernetes Service).

1. **Instance RDS pour la base de données PostgreSQL :**

* **Description :** Une instance Amazon RDS est utilisée pour héberger la base de données PostgreSQL. RDS simplifie l'administration de la base de données en automatisant des tâches telles que les sauvegardes, les mises à jour de version et la gestion des réplications.
* **Bonne Pratique :** Configuration de RDS pour le chiffrement des données au repos, la sauvegarde régulière, et la configuration des paramètres de performance pour répondre aux exigences de l'application.

# Sécurité

Contrôles concernant la sécurité de l'architecture

1. **Mesures de Sécurité :**

* **Hachage des Mots de Passe et Données Sensibles :** Les mots de passe et autres informations sensibles sont stockés dans la base de données après avoir été hachés à l'aide d'algorithmes sécurisés comme BCrypt. Cela garantit que même en cas de compromission de la base de données, les données restent sécurisées.
* **Exigence de Composition de Mot de Passe Robuste :** Les utilisateurs sont tenus de créer des mots de passe robustes qui répondent à des critères stricts de complexité, incluant une combinaison de lettres majuscules et minuscules, chiffres et caractères spéciaux.
* **Authentification à 2 facteurs :** Après avoir saisi leur mot de passe, les utilisateurs doivent fournir un code unique généré via une application ou envoyé par email ou SMS.
* **Gestion des Sessions :** Utilisation de Spring Security pour gérer de manière sécurisée les sessions utilisateur, avec des mécanismes de gestion de session stateful ou stateless selon les besoins de l'application.

1. **Spring Security :**

* **Description :** Spring Security est utilisé pour sécuriser l'application à différents niveaux, notamment l'authentification, l'autorisation et la protection contre les vulnérabilités communes.

1. **Load Balancer pour Gérer les Attaques DDoS :**

* **Description :** Un load balancer, tel qu'AWS Elastic Load Balancer (ELB), est configuré pour distribuer le trafic entrant de manière équilibrée entre les instances EC2 du front-end et du back-end.
* **Mesures de Sécurité :** Configuration d'ELB pour détecter et atténuer les attaques DDoS en filtrant le trafic malveillant tout en maintenant la disponibilité des services pour les utilisateurs légitimes.

1. **Échanges FE-BE en HTTPS :**

* **Description :** Tous les échanges de données entre le front-end (FE) et le back-end (BE) sont sécurisés via HTTPS.
* **Mesures de Sécurité :**
  + Utilisation de certificats SSL/TLS pour chiffrer le trafic HTTP afin de prévenir les attaques de type Man-in-the-Middle (MITM), assurant ainsi la confidentialité et l'intégrité des données échangées.

1. **Sécurité des WebSockets :**

* **Description :** La sécurité des WebSockets est cruciale pour protéger les données échangées en temps réel entre le front-end (FE) et le back-end (BE). L'authentification est réalisée à l'aide de tokens JWT (JSON Web Tokens) lors de l'établissement de la connexion WebSocket.
* **Mesures de Sécurité :**
  + Utilisation de WSS (WebSocket Secure) pour chiffrer les communications, empêchant ainsi les interceptions.
  + Validation des données entrantes pour prévenir les attaques, comme les injections ou les détournements de sessions WebSocket.
  + Gestion des événements de connexion et de déconnexion pour assurer la stabilité et la sécurité de la communication.

